1. Les docstrings

1.1 C'est quoi ?

Une **docstring** est une chaîne de caractères placée :

- au début d'un **module** (fichier),
- d'une **fonction**,

...et utilisée comme documentation intégrée (accessible via `help()` et `.__doc__`).

✅ Docstring = documentation + contrat d'usage + exemples.

1.2 Syntaxe

On utilise généralement des triples guillemets :

```
def aire_cercle(r):
    """Calcule l'aire d'un cercle de rayon r."""
    return 3.14159 * r * r
```

Accès :

```
print(aire_cercle.__doc__)
help(aire_cercle)
```

1.3 Contenu conseillé

Une docstring utile contient souvent :

1. **Une phrase courte** (ce que fait la fonction).
2. **Paramètres** (type, signification).
3. **Retour** (type, signification).
4. **Exceptions** possibles.
5. **Exemples** (souvent au format doctest).

Exemple simple mais propre :

```
def diviser(a, b):
    """
    Divise a par b.

    Args:
        a (float): numérateur
        b (float): dénominateur
```

```
    Returns:
        float: résultat de a / b

    Raises:
        ZeroDivisionError: si b == 0
    """
    return a / b
```

> Note : il existe plusieurs styles (Google, NumPy, reST/Sphinx). L'important est d'être **cohérent** dans un projet.

---

2. Les `assert`

2.1 À quoi ça sert ?

`assert` sert à **détecter rapidement** une condition qui devrait être vraie pendant le développement.

```
assert condition, "message si faux"
```

Si la condition est fausse → Python lève `AssertionError`.

2.2 Exemples

Vérifier des préconditions

```
def inverse(x):
    assert x != 0, "x doit être >= 0"
    return 1/x
```

Vérifier des invariants internes

```
def moyenne(notes):
    assert len(notes) > 0
    m = sum(notes) / len(notes)
    assert 0 <= m <= 20
    return m
```

2.3 Bonnes pratiques

Pour du “vrai” contrôle d’erreur en prod : lève des exceptions explicites :

```
def racine(x):
    if x < 0:
        raise ValueError("x doit être >= 0")
    return x ** 0.5
```

---

3. `doctest` : tester via les docstrings

3.1 Le principe

`doctest` exécute des **exemples écrits dans la docstring** au format REPL (>>>) et vérifie que la sortie correspond.

Exemple :

```
def somme(a, b):
    """
    Retourne a + b.

    >>> somme(2, 3)
    5
    >>> somme(-1, 1)
    0
    """
    return a + b
```

3.2 Lancer doctest

```
if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

3.3 Points importants (pièges fréquents)

A. Les flottants

Les flottants peuvent poser problème (arrondis). Préfère arrondir :

```
def ratio(a, b):
    """
```

```
    >>> ratio(1, 3)
    0.33333333333333333333
    """
    return a / b
```

B. Tester des exceptions

On peut vérifier qu'une exception est levée :

```
def inverser(x):
    """
    >>> inverser(2)
    0.5
    >>> inverser(0)
    Traceback (most recent call last):
    ...
    ZeroDivisionError: division by zero
    """
    return 1 / x
```

Les `...` servent de joker pour des parties variables.

3.4 Quand utiliser doctest ?

✅ Excellent pour :

- des fonctions "pures" (entrées → sorties),
- documenter des exemples d'usage,
- éviter que la doc ne devienne fausse.

---

4. Trio gagnant : docstring + assert + doctest (exemple complet)

```
def factorielle(n):
    """
    Calcule n! (factorielle) pour un entier n >= 0.

    Args:
        n (int): entier >= 0

    Returns:
        int: n!
```

```
    Examples:
        >>> factorielle(0)
        1
        >>> factorielle(5)
        120
        >>> factorielle(3)
        6
    """
    assert isinstance(n, int), "n doit être un int"
    assert n >= 0, "n doit être >= 0"

    res = 1
    for k in range(2, n + 1):
        res *= k
    return res


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```